Traffic Light Simulation

T.K.SUPRIYA - 21B01A54A9 - AIDS
M.RAMA SATYA DEVI - 21B01A0225 - EEE
D.NEHA - 21B01A1241 - IT
R.VIDYA - 21B01A12F8 - IT
T.N.M.VSAILUSHA - 21B01A54B0 - AIDS

SVECW

March 25,2023

Introduction

It contains a 4-way traffic intersection with traffic signals controlling the flow of traffic in each direction. Each signal has a timer on top of it which shows the time remaining for the signal to switch from green to yellow, yellow to red, or red to green. Vehicles such as cars, bikes, buses, and trucks are generated, and their movement is controlled according to the signals and the vehicles around them.

Approach

- This is the code for simulation of traffic management in a 4-Way intersection.It uses the pygame library to create a GUI(graphical user interface).
- The code initializes the pygame library,creates classes for the traffic signal and vehicles,and defines variables for vehicle properties and the stop line.
- The traffic class defines the red,yellow and green states of the signal.The vehicle class is a subclass of pygame's sprite class(The Sprite class is intended to be used as a base class for the different types of objects in the game.) and contains the properties of a vehicle like speed,lane,direction and a stopping point.
- The vehicles are stored in dictionary with respect to directions as keys and list of vehicles as values.
- The code sets the starting and stopping points of each vehicles based on the previous vehicles in the same lane.The GUI is updated with the render method of the vehicle class.

Learnings

- **import random** - used to generate random numbers
- **import time** - to generate time
- **import threading** - used to run multiple threads(Tasks,Function calls) at the same time
- **import pygame** - development of multimedia applications like video games using python
- **import sys** - manipulate different parts of the python runtime environment

Challenges

- **Modeling real-world traffic:** Accurately simulating the complex and unpredictable nature of real-world traffic is difficult.
- **Representing all road users:** Simulating all road users, including vehicles, bicycles, pedestrians, and public transport, is challenging and requires accurate representation of their behavior and interactions.
- **Dealing with non-linearity:** Traffic light systems are inherently non-linear and dynamic, making it difficult to model and simulate.
- **Managing real time-data:** Incorporating real-time data from sensors,cameras,and other sources into traffic light simulations can be challenging and requires efficient data processing and management

Statistics

- **Number of lines of code**:230
- **Functions**:

* **def render(self,screen)** - to display the image on the screen
* **def move(self)** - to control the movement of vehicles according to the traffic light and the vehicles ahead
* **def initialize()** - initialization of signals with default values
* **def repeat()** - the function repeat() is called at the end of the initialize() function above is a recursive function that runs our entire simulation.This is the driving force of our simulation
* **def updateValues()** - update values of the signal timers after every second
* **def generateVehicles()** - generating the vehicles in simulation

Demo/Screen shots

graphics

```
import random
import time
import threading
import pygame
import sys

defaultGreen = {0:10, 1:10, 2:10, 3:10}
defaultRed = 150
defaultYellow = 5

signals = []
noOfSignals = 4
currentGreen = 0
nextGreen = (currentGreen+1)%noOfSignals
currentYellow = 0

speeds = {'car':2.25, 'bus':1.8, 'truck':1.8, 'bike':2.5}


x = {'right':[0,0,0], 'down':[755,727,697], 'left':[1400,1400,1400], 'up':[602,627,657]}
y = {'right':[348,370,398], 'down':[0,0,0], 'left':[498,466,436], 'up':[800,800,800]}

vehicles = {'right': {0:[], 1:[], 2:[], 'crossed':0}, 'down': {0:[], 1:[], 2:[], 'crossed':0},
            'left': {0:[], 1:[], 2:[], 'crossed':0}, 'up': {0:[], 1:[], 2:[], 'crossed':0}}
vehicleTypes = {0:'car', 1:'bus', 2:'truck', 3:'bike'}
directionNumbers = {0:'right', 1:'down', 2:'left', 3:'up'}
```

image1

Demo/Screen shots

graphics

```
signalCoods = [(530,230),(810,230),(810,570),(530,570)]
signalTimerCoods = [(530,210),(810,210),(810,550),(530,550)]


stopLines = {'right': 590, 'down': 330, 'left': 800, 'up': 535}
defaultStop = {'right': 580, 'down': 320, 'left': 810, 'up': 545}

stoppingGap = 15
movingGap = 15

pygame.init()
simulation = pygame.sprite.Group()

class TrafficSignal:
    def _init_(self, red, yellow, green):
        self.red = red
        self.yellow = yellow
        self.green = green
        self.signalText = ""

class Vehicle(pygame.sprite.Sprite):
    def _init_(self, lane, vehicleClass, direction_number, direction):
        pygame.sprite.Sprite._init_(self)
        self.lane = lane
        self.vehicleClass = vehicleClass
        self.speed = speeds[vehicleClass]
        self.direction_number = direction_number
```

image2

Demo/Screen shots

graphics

```
self.direction = direction
self.x = x[direction][lane]
self.y = y[direction][lane]
self.crossed = 0
vehicles[direction][lane].append(self)
self.index = len(vehicles[direction][lane]) - 1
path = "images/" + direction + "/" + vehicleClass + ".png"
self.image = pygame.image.load(path)

if(len(vehicles[direction][lane])>1 and vehicles[direction][lane][self.index-1].crossed==0):
    if(direction=='right'):
        self.stop = vehicles[direction][lane][self.index-1].stop - vehicles[direction][lane][self.index-1].image.get_rect().widt
    elif(direction=='left'):
        self.stop = vehicles[direction][lane][self.index-1].stop + vehicles[direction][lane][self.index-1].image.get_rect().widt
    elif(direction=='down'):
        self.stop = vehicles[direction][lane][self.index-1].stop - vehicles[direction][lane][self.index-1].image.get_rect().heig
    elif(direction=='up'):
        self.stop = vehicles[direction][lane][self.index-1].stop + vehicles[direction][lane][self.index-1].image.get_rect().heig
else:
    self.stop = defaultStop[direction]

if(direction=='right'):
    temp = self.image.get_rect().width + stoppingGap
    x[direction][lane] -= temp
elif(direction=='left'):
    temp = self.image.get_rect().width + stoppingGap
    x[direction][lane] += temp
```

image3

Demo/Screen shots

graphics

```
    elif(direction=='down'):
        temp = self.image.get_rect().height + stoppingGap
        y[direction][lane] -= temp
    elif(direction=='up'):
        temp = self.image.get_rect().height + stoppingGap
        y[direction][lane] += temp
    simulation.add(self)

def render(self, screen):
    screen.blit(self.image, (self.x, self.y))

def move(self):
    if(self.direction=='right'):
        if(self.crossed==0 and self.x+self.image.get_rect().width>stopLines[self.direction]):
            self.crossed = 1
        if((self.x+self.image.get_rect().width<=self.stop or self.crossed == 1 or (currentGreen==0 and currentYellow==0)) and (se
            self.x += self.speed
    elif(self.direction=='down'):
        if(self.crossed==0 and self.y+self.image.get_rect().height>stopLines[self.direction]):
            self.crossed = 1
        if((self.y+self.image.get_rect().height<=self.stop or self.crossed == 1 or (currentGreen==1 and currentYellow==0)) and (s
            self.y += self.speed
    elif(self.direction=='left'):
        if(self.crossed==0 and self.x<stopLines[self.direction]):
            self.crossed = 1
        if((self.x>=self.stop or self.crossed == 1 or (currentGreen==2 and currentYellow==0)) and (self.index==0 or self.x>(vehic
            self.x -= self.speed
```

image4

Demo/Screen shots

graphics

```
        elif(self.direction=='up'):
            if(self.crossed==0 and self.y<stopLines[self.direction]):
                self.crossed = 1
            if((self.y>=self.stop or self.crossed == 1 or (currentGreen==3 and currentYellow==0)) and (self.index==0 or self.y>(v
                self.y -= self.speed

def initialize():
    ts1 = TrafficSignal(0, defaultYellow, defaultGreen[0])
    signals.append(ts1)
    ts2 = TrafficSignal(ts1.red+ts1.yellow+ts1.green, defaultYellow, defaultGreen[1])
    signals.append(ts2)
    ts3 = TrafficSignal(defaultRed, defaultYellow, defaultGreen[2])
    signals.append(ts3)
    ts4 = TrafficSignal(defaultRed, defaultYellow, defaultGreen[3])
    signals.append(ts4)
    repeat()

def repeat():
    global currentGreen, currentYellow, nextGreen
    while(signals[currentGreen].green>0):
        updateValues()
        time.sleep(1)
    currentYellow = 1

    for i in range(0,3):
        for vehicle in vehicles[directionNumbers[currentGreen]][i]:
            vehicle.stop = defaultStop[directionNumbers[currentGreen]]
```

image5

Demo/Screen shots

graphics

```
    while(signals[currentGreen].yellow>0):
        updateValues()
        time.sleep(1)
    currentYellow = 0

    signals[currentGreen].green = defaultGreen[currentGreen]
    signals[currentGreen].yellow = defaultYellow
    signals[currentGreen].red = defaultRed

    currentGreen = nextGreen
    nextGreen = (currentGreen+1)%noOfSignals
    signals[nextGreen].red = signals[currentGreen].yellow+signals[currentGreen].green
    repeat()

def updateValues():
    for i in range(0, noOfSignals):
        if(i==currentGreen):
            if(currentYellow==0):
                signals[i].green-=1
            else:
                signals[i].yellow-=1
        else:
            signals[i].red-=1

def generateVehicles():
    while(True):
        vehicle_type = random.randint(0,3)
```

image6

Demo/Screen shots

graphics

```
        lane_number = random.randint(1,2)
        temp = random.randint(0,99)
        direction_number = 0
        dist = [25,50,75,100]
        if(temp<dist[0]):
            direction_number = 0
        elif(temp<dist[1]):
            direction_number = 1
        elif(temp<dist[2]):
            direction_number = 2
        elif(temp<dist[3]):
            direction_number = 3
        Vehicle(lane_number, vehicleTypes[vehicle_type], direction_number, directionNumbers[direction_number])
        time.sleep(1)

class Main:
    thread1 = threading.Thread(name="initialization",target=initialize, args=())
    thread1.daemon = True
    thread1.start()

    black = (0, 0, 0)
    white = (255, 255, 255)

    screenWidth = 1400
    screenHeight = 800
    screenSize = (screenWidth, screenHeight)
```

image7

Demo/Screen shots

graphics

```
    background = pygame.image.load('images/intersection.png')

    screen = pygame.display.set_mode(screenSize)
    pygame.display.set_caption("SIMULATION")

    redSignal = pygame.image.load('images/signals/red.png')
    yellowSignal = pygame.image.load('images/signals/yellow.png')
    greenSignal = pygame.image.load('images/signals/green.png')
    font = pygame.font.Font(None, 30)

    thread2 = threading.Thread(name="generateVehicles",target=generateVehicles, args=())
    thread2.daemon = True
    thread2.start()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        screen.blit(background,(0,0))
        for i in range(0,noOfSignals):
            if(i==currentGreen):
                if(currentYellow==1):
                    signals[i].signalText = signals[i].yellow
                    screen.blit(yellowSignal, signalCoods[i])
                else:
                    signals[i].signalText = signals[i].green
```

image8

Demo/Screen shots

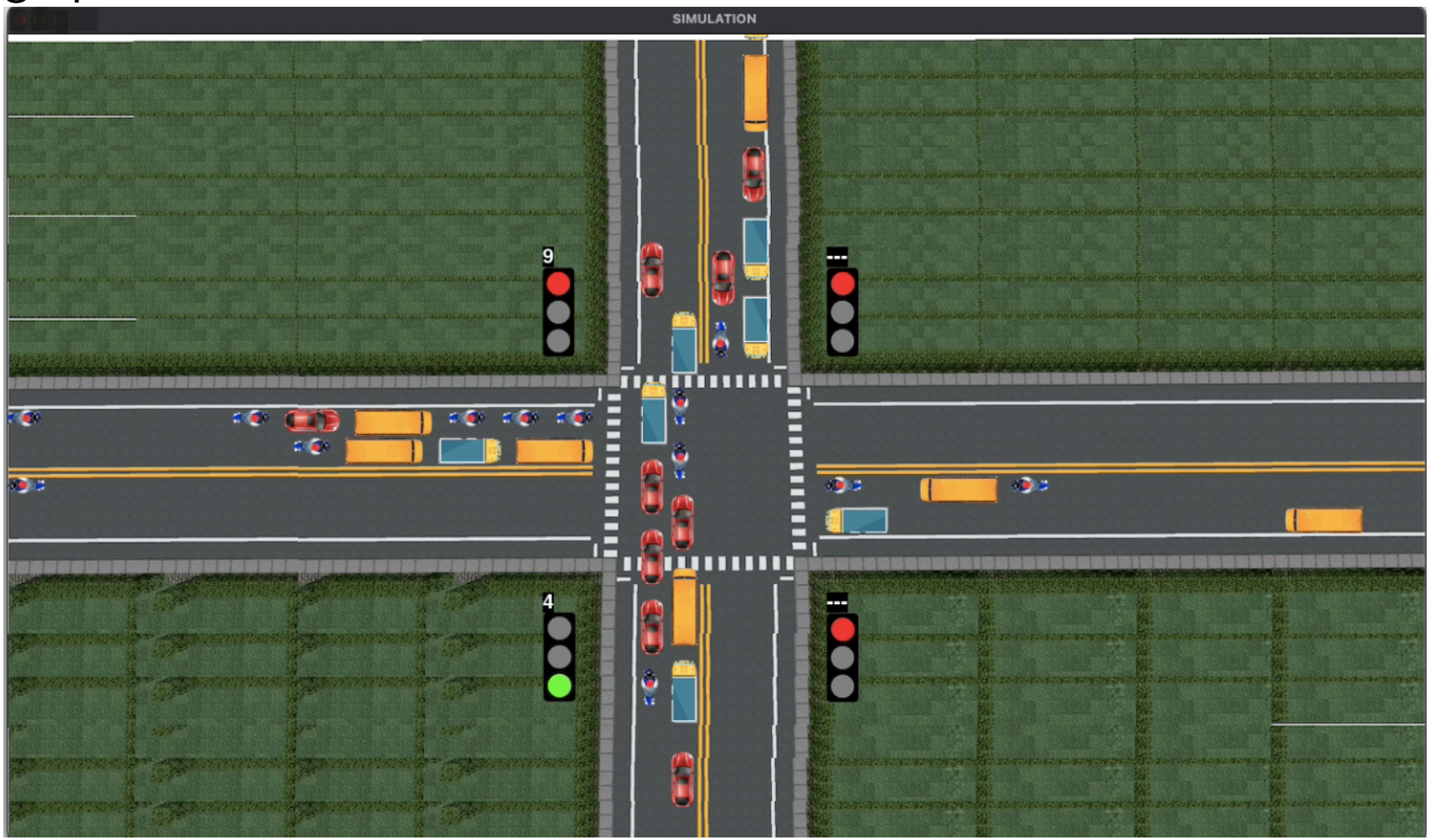
graphics

```
                    screen.blit(greenSignal, signalCoods[i])
            else:
                if(signals[i].red<=10):
                    signals[i].signalText = signals[i].red
                else:
                    signals[i].signalText = "---"
                screen.blit(redSignal, signalCoods[i])
        signalTexts = ["","","",""]

        for i in range(0,noOfSignals):
            signalTexts[i] = font.render(str(signals[i].signalText), True, white, black)
            screen.blit(signalTexts[i],signalTimerCoods[i])

        for vehicle in simulation:
            screen.blit(vehicle.image, [vehicle.x, vehicle.y])
            vehicle.move()
        pygame.display.update()


Main()
```

image9

Demo/Screen shots

graphics



output

THANK YOU